Rockin' Windoze in the Free (Linux) World

Mike Fienen, USGS WiWSC/CIDA-EMU

March 5, 2012

Preliminaries

Condor is developed in Linux and made available in Windows with some limitations on functionality (specifically the ability to SSH to a running job). Also, Windows licensing can be prohibitively expensive. But....beoPEST is the opposite—developed on Windows with some features lagging a bit behind for Linux. The user base for many of our modeling projects is working in Windows and sometimes includes custom or proprietary codes that would be difficult or tedious to compile and optimize for Linux. The bottom line is, it may be advantageous to run Condor on Linux platforms but run the underlying models and beoPEST in Windows.

WINE is an open-source project that makes it possible to accomplish the goal stated above of running Windows binaries on a Linux platform. WINE stands for Wine is Not an Emulator (http://www.winehq.org). In this short note, I outline the steps required to port a Windows project to WINE on Linux using Condor.

Installation

In the CIDA-EMU Linux pool, installation of WINE is very straightforward. One must log in as root and run the command `yum install wine` accept a couple queries about download size, and the installation should work.

A key dependency on Scientific Linux 6.x (which we are running—it's a RedHat clone) is EPEL (Extra Packages for Enterprise Linux). This is already in place on the CIDA-EMU machines, but if it's missing on a machine, it can be installed by typing the following as root:

```
rpm -i http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-5.noarch.rpm
```

Once this is successfully done, you can run the yum install command above and all is well!

Running Wine in Condor

A key issue with running WINE in Condor is related to permissions–by default, WINE needs to write to a folder it creates at the user directory. Since we install as root, this location is `/`. For security reasons, one cannot run Condor as root, nor can Condor write to `/` so an alternate location for WINE to write must be established. Luckily, the environment variable `WINEPREFIX` specifies this writing location. Condor also creates a scratch directory for each run, so no toes are stepped on as multiple runs waltz away on a single machine.

To force WINE to use this scratch directory, the following line should be in the shell script that Condor calls on each machine:

```
export WINEPREFIX$_CONDOR_SCRATCH_DIR
```

After that step, all other Windows applications are run as they normally would be on the command line or in a batch file, except they must be preceded by the word "wine". An example is:

```
wine beopest64.exe my_sweet_project.pst /h /p1 host:port
```

In this paradigm, mixing Linux and Windows commands within a single shell script is easily accomplished. Following is an example of a shell script called by Condor to start a slave in beoPEST:

```
#!/bin/sh
host$1
port$2
export WINEPREFIX$_CONDOR_SCRATCH_DIR
wine unzip Python27.zip
tar xzf data.tar
cd data
wine beopest64 my_sweet_model.pst /h /p1 $host:$port
```

Parting Thoughts/Notes

By running Condor in Linux with Windows floating in underneath, a couple very important features are open. One is tailing files. By opening a terminal window, one can type `tail -f worker_121_0.out` to dynamically follow the screen output of the slave running as run 121 (according to Condor).

Another feature is the command `condor_ssh_to_job <job#>`, which floats in on the Condor user allowing one to tail, secure copy (`scp`) or otherwise interact with a running model.

Batch files that are called by beoPEST cascade their dependencies, meaning that the batch file needs not be altered (in other words, the word `wine` needs not be prepended to program calls in the batch file. An exception to this is anything that requires changing the system path (on the Windows side). While WINE supports a registry editing GUI, I have not yet figured out how to programmatically change the PATH variable, for example. An example of this issue is if Python is shipped along with the model files (as indicated above). The python directory gets unzipped and on native windows implementations, the path is updated with the location of this sandboxed python installation. This is not easily done in WINE, but the only implication is that the batch file where python is called must be updated. In the example above, the line in the batch file `python myscript.py` must simply be changed to `..\python27\python myscript.py`. In other words, the path must be fully qualified.